**NOTE**: Like the last project, it is very important that if you are not working on the project or after finishing the project, you must **STOP** or **DESTROY** the instance to avoid future charging.

In this project, we will process a large corpus of movie ratings for providing recommendations. When you're done, your program will help you decide what to watch on Netflix tonight. For each pair of movies in the data set, you will compute their [statistical correlation](http://en.wikipedia.org/wiki/Correlation_and_dependence) and [cosine similarity](http://en.wikipedia.org/wiki/Cosine_similarity) (see [this blog](http://blog.echen.me/2012/02/09/movie-recommendations-and-more-via-mapreduce-and-scalding/) for a discussion of these and other potential similarity metrics). Since this isn't a statistics class, the calculation of similarity metrics for Python and Java will be provided, but you need to provide them with the correct inputs.

**Data**

We have two input data sets: a [small set](http://files.grouplens.org/datasets/movielens/ml-latest-small.zip) for testing on your local machine or on cloud and a [large set](http://files.grouplens.org/datasets/movielens/ml-1m.zip) for running on AWS or GCP. More info about the data set can be found [here](https://grouplens.org/datasets/movielens/). If you are using VM instances, please use “wget” and “unzip” commands to download and unzip the files.

For both data sets, you will find two input files:

* **movies.csv** or **movies.dat** contains identification and genre information for each movie.

Lines are of the form:

*MovieID,Movie Title,Genre*

0068646,The Godfather (1972),Crime|Drama

0181875,Almost Famous (2000),Drama|Music

* **ratings.csv** or **ratings.dat** contains a series of individual user movie ratings.

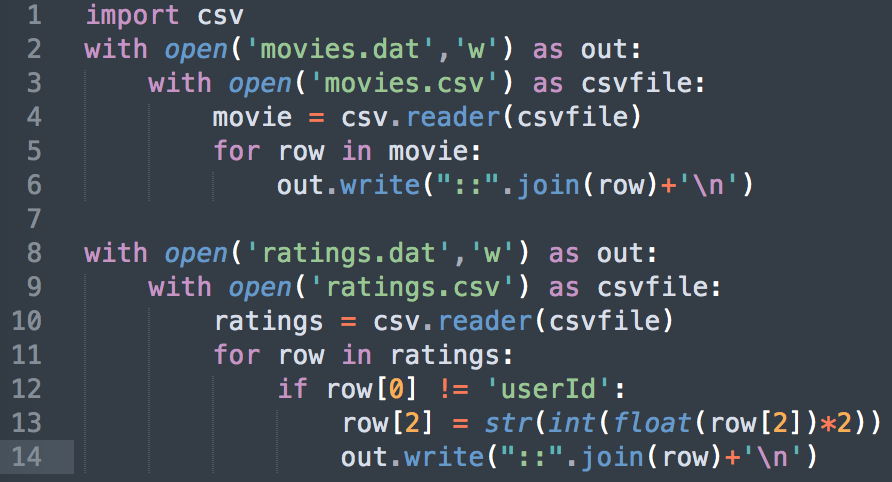
Lines are of the form:

*UserID,MovieID,Rating,:Timestamp*

120,0068646,10,1365448727

374,0181875,9,1374863640

NOTE: to convert the csv files to dat files, use following Python code:



In addition to these two input files, the program should take a few additional arguments:

* **-m [movie title]:** The title of a movie for which we'd like to see similar titles. You should be able to accept multiple movie titles with more than one **-m** argument.
* **-k [number of items]:** For each of the movies specified using **-m**, this specifies how many of the top matches we'd like to see. In other words, running with "-m The Godfather (1972) -k 10" would be asking for "the top ten movie recommendations if you liked The Godfather." (Default 15)
* **-l [similarity lower bound]:** When computing movie similarity metrics, we'll produce a floating-point value in the range [-1, 1]. This input says to ignore any movie parings whose similarity metric is below this value. (Default 0.4)
* **-p [minimum rating pairs]:** When computing similarity metrics, ignore any pair of movies that don't have at least this many shared ratings. (Default 8)

**Please don't attempt to filter down to the movies specified via -m until the final step. I want you to compute the similarities for all movies. The -m argument is there to reduce the output size and make reading (and grading) the output easier.** For the other arguments (-k, -l, and -p), you may filter whenever you want.

**Output**

Since we're computing two similarity metrics, we'll need to combine them into a single similarity value somehow. For your submission, you should blend the values together, using 50% of each. That is, your final value for a pair of movies is 0.5 \* statistical correlation + 0.5 \* the cosine correlation for the pair.

For each movie selected (-m), sort them from largest to smallest by their blended similarity metric, outputting only the top K (-k) most similar movies that are have at least the minimum blended similarity score (-l). For movies meeting this criterion, you should output:

* The name of the movie for which we want similar titles (specified via -m).
* The name of a similar movie.
* The blended similarity metric that these two movies share.
* The statistical correlation that these two movies share.
* The cosine correlation that these two movies share.
* The number of ratings this pair of movies had in common.

You may format your output however you like, as long as the values are in the correct order and I can reasonably make sense of it by looking at it briefly.

**Steps**

You may structure your sequence of map/reduce tasks at your choice. However, I recommend the following sequence of steps:

1. **Join the input files:** Initially, you have two input files (ratings.dat and movies.dat). You'll get most of the important info from ratings.dat, but it only has movie IDs rather than movie names. For the first step, you can assign names to the rated movies and drop the movie ID. This way, you can refer to movies by their name going forward. You probably want your reducer's output to be key: user id, value: (movie title, rating) (which you will use as the input of next map reduce step). Hint: ratings file has 4 items each line, while movies file has 3 each line. This difference can be used to differ input.
2. **Produce movie rating pairs:** Next, you want to organize the movies into pairs, recording the ratings of each when **a user has rated both movies (i.e. for each user, you will create pairs for every two movies)**. This gives you vectors to use for the similarity metrics. For example, suppose we have three users, Alice, Bob, and Charlie:

Alice has rated Almost Famous a 10, The Godfather a 9, and Anchorman a 4.

Bob has rated Almost Famous a 7 and Anchorman a 10.

Charlie has rated The Godfather a 10 and Anchorman an 8.

You would end up with records that look like:

Key: (Almost Famous, The Godfather)

Values: (10, 9)

Key: (Almost Famous, Anchorman)

Values: (10, 4), (7, 10)

Key: (Anchorman, The Godfather)

Values: (4, 9), (8, 10)

1. **Protip:** You'll want to ensure that the keys you output are consistent for a pair of movies, for example, by putting them in alphabetical order. Otherwise, you run the risk of having two keys for a pair of movies, e.g., (Anchorman, The Godfather) and (The Godfather, Anchorman). Having more than one key for a pair is bad, as they will be treated independently (and probably sent to different machines for processing).
2. **Compute the similarity scores:** Given keys that tell you a pair of movie names and values that contain a sequence of pairs, each corresponding to how a user rated that pair, you now have the information you need to compute similarity scores for that pair of movies. You will need to organize the data before the calculation. For example:

Movies “Anchorman” and “The Godfather” has 5 values: (4, 9), (8, 6), (4, 9), (8, 4), (7, 9), the input for the calculation will be [4, 8, 4, 8, 7] and [9, 6, 9, 4, 9]. Then you put them into the calculation script.

**In Python:**

The statistical calculation in Python is as follow. You may want to add the calculation process



**In Java:**

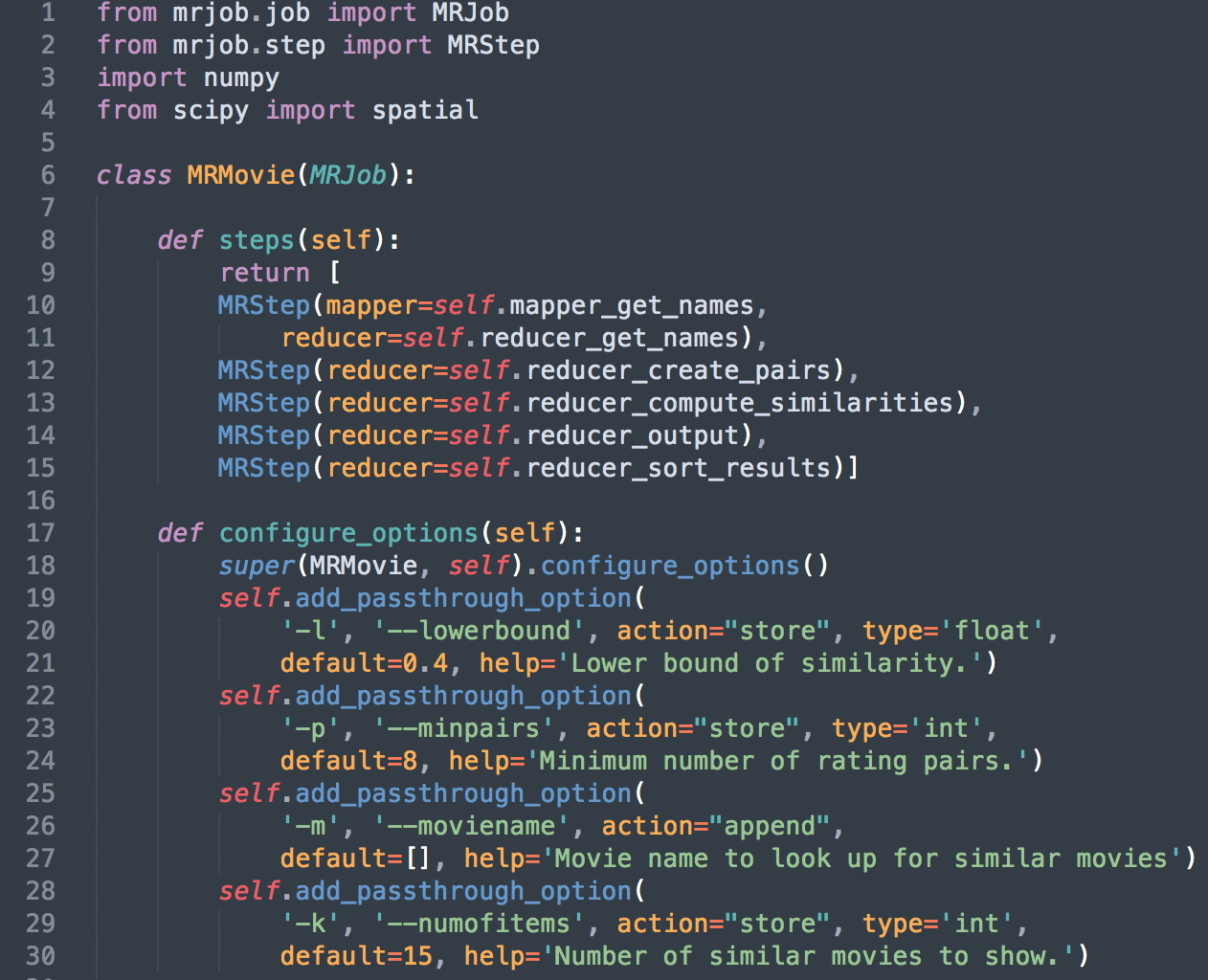
For two list of numbers r1 and r2, you can refer [here](http://stackoverflow.com/questions/28428365/how-to-find-correlation-between-two-integer-arrays-in-java) for statistical correlation and [here](http://stackoverflow.com/questions/520241/how-do-i-calculate-the-cosine-similarity-of-two-vectors) for cosine correlation.

1. **Filter and format the output:** Filter out the movies that weren't specified with the -m flag and sort the output by similarity metric so that it conforms to the desired output format.

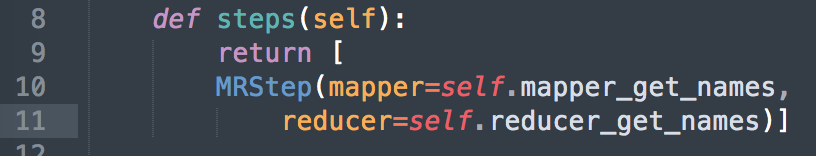
**Python code breakdown**

Since most of us are new to the Python “mrjob”, a sample code is provided with several steps to be filled in.

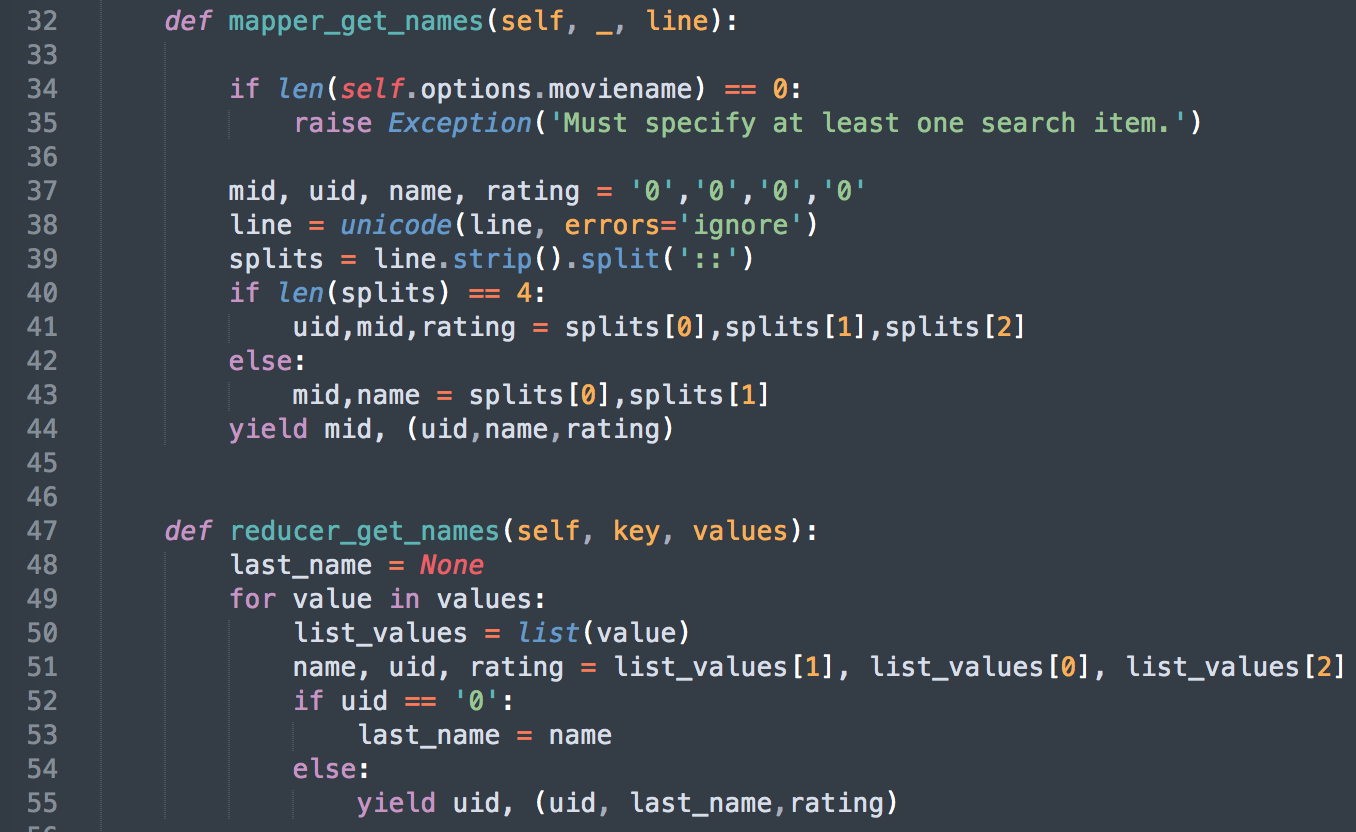
* Defining the steps and command line argumentations parsing are already provided:



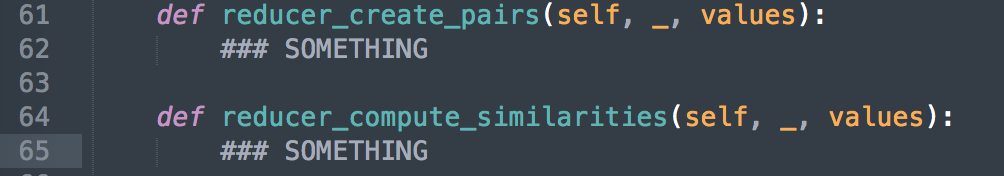
* To check the results of each step or debug your code, you can only give the function part of the steps (this example will print the results of the first step):



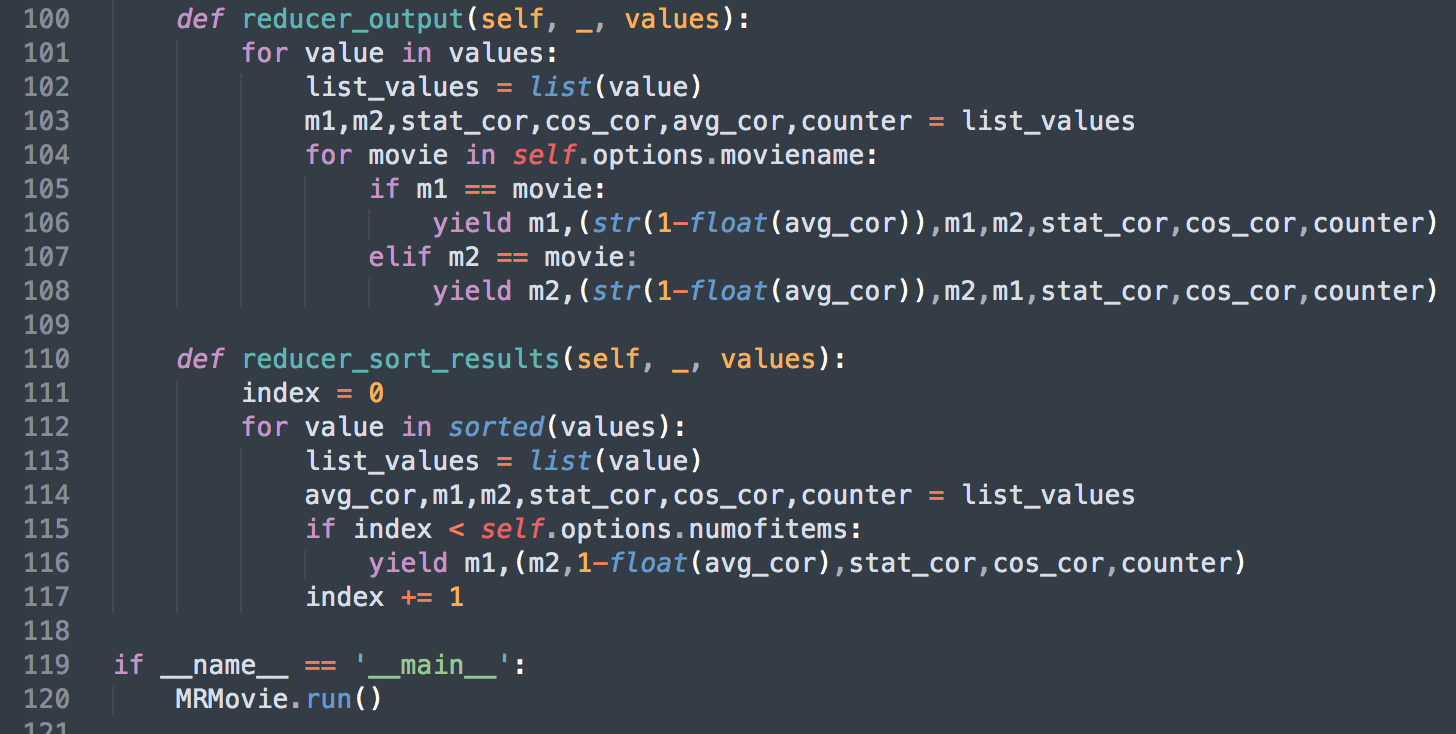
* The first step is to get the user ID, movie name, and ratings from the input files, and use the user ID to group them (group by key is the feature of map reduce). The code is also provided, so you can understand the input and output of mrjob.



* In the second and third steps, you should create the rating pairs and compute the similarities. Please fill in the function with proper codes. These are the only parts of the project that need you to program.



* The fourth and last steps deals with the output:



* The script should be saved as “MRMovie.py”. The next section will describe how to run the script.

**Runners**

For the small data set, you can run it locally or directly on an AWS/GCP instance. It will probably take a few minutes to complete. You *can* run over the large data set locally too, if you want, but it will take at least a few hours and your machine may crash. Instead, let's farm it out to AWS or GCP with multiple instances configured.

The tutorial of the “mrjob” can be found [here](https://pythonhosted.org/mrjob/index.html). It comes with the links to run this on [Amazon EMR](https://aws.amazon.com/documentation/emr/) or [Google Cloud](https://cloud.google.com/dataproc/docs/).

To run on Amazon, you'll need to tell MRJob to use "emr" as its runner. You'll also need to give it some basic configuration information. Edit your mrjob.conf with the following contents (more information [here](https://pythonhosted.org/mrjob/guides/configs-basics.html)):

runners:

inline:

base\_tmp\_dir: /local

emr:

core\_instance\_type: t2.micro ###use a faster instance if you get education credit from Amazon (refer [here](https://aws.amazon.com/ec2/pricing/on-demand/) for more instance info)

num\_core\_instances: 4 ###use more instances if you want it to be solved faster

aws\_access\_key\_id: [your access key]

aws\_secret\_access\_key: [your secret key]

aws\_region: us-east-2a ###can be other available zones

Attention: YAML doesn't allow tabs, so only use space in mrjob.conf; by using m3.xlarge and 5 instances, you should be able to solve the large set within a hour.

Other than use security key sets, you can also use AWS’s IAM to configure your account. Now, you should be able to invoke MRJob with "--conf-path mrjob.conf -r emr" to run your code in the cloud.

Similarly, you can run it with GCP [Dataproc](https://pythonhosted.org/mrjob/guides/runners.html#running-on-dataproc) by editing the “mrjob.conf”.

**Deliverables:**

You need to include in your report the following information:

* The source code that you have written
* Run you script with at least 3 movies and provide the screenshot of the result. Here is an example of executing the code and the last output.

